1. Connected Components

int vis[100005];

vector<int> ar[100005];

void clean(int n){

    for(int i=0; i<=n; ++i){

        ar[i].clear();

        vis[i]=0;

    }

}

void dfs(int v){

    vis[v]=1;

    for(int child: ar[v]){

        //int child=ar[v][j];

        if(vis[child]==0) dfs(child);

    }

}

int Solution::solve(int A, vector<vector<int> > &B) {

    clean(A);

    int m=B.size();

    for(int i=0;i<m;i++) {

        ar[B[i][0]].push\_back(B[i][1]);

        ar[B[i][1]].push\_back(B[i][0]);

    }

    int cc=0;

    for(int i=1;i<=A;i++){

        if(vis[i]==0) {dfs(i);cc++;}

    }

    return cc;

}

1. Level Order

void build(TreeNode\* root,vector<vector<int>> &ret,int depth){

if(!root) return;

if(ret.size()==depth) ret.push\_back(vector<int>(0));

ret[depth].push\_back(root->val);

build(root->left,ret,depth+1);

build(root->right,ret,depth+1);

return;

}

vector<vector<int> > Solution::levelOrder(TreeNode\* B) {

vector<vector<int>> ret;

build(B,ret,0);

return ret;

}

1. Cycle in a undirected graph

int visited[1000];

int cycle;

void DFS(vector <vector<int> > adj, int i, vector<int> ancestry, vector<vector<int> > &B)

{

visited[i]=1;

ancestry.push\_back(i);

for(int j=0; j<adj[i].size(); j++)

{

if(!visited[adj[i][j]])

{

DFS(adj, adj[i][j], ancestry, B);

}

if(find(ancestry.begin(), ancestry.end(), adj[i][j])!=ancestry.end() && (adj[i][j]!=ancestry[ancestry.size()-2]))

{

cycle=1;

}

}

}

int Solution::solve(int A, vector<vector<int> > &B)

{

if(A>1000)

{

return 1;

}

int i, j;

vector <vector<int> > adj(A+1);

for(i=0; i<B.size(); i++)

{

adj[B[i][0]].push\_back(B[i][1]);

adj[B[i][1]].push\_back(B[i][0]);

}

memset(visited, 0, sizeof(visited));

cycle=0;

for(i=1; i<=A; i++)

{

if(!visited[i])

{

// vector<int> depth\_first;

DFS(adj, i, {}, B);

}

}

return cycle;

}

1. Cycle in a directed graph

bool findCycle(int cur, vector<vector<int>>&adj, vector<bool>&visited, vector<bool>&recVisited){

visited[cur] = true;

recVisited[cur] = true;

for(auto itr: adj[cur]){

if(!visited[itr]){

if(findCycle(itr, adj, visited, recVisited))return true;

}

// if it is visited then check in recVisited

else if(recVisited[itr]){

return true;

}

}

recVisited[cur] = false;

return false;

}

int Solution::solve(int v, vector<vector<int> > &B) {

vector<vector<int>>adj(v+1);

for(int i = 0; i < B.size(); ++i){

adj[B[i][0]].push\_back(B[i][1]);

}

vector<bool>visited(v+1);

vector<bool>recVisited(v+1);

for(int i = 1; i <= v; ++i){

if(!visited[i]){

if(findCycle(i, adj, visited, recVisited)){

return true;

}

}

}

return false;

}

1. Path in a directed graph

vector<vector<int>> adj;

vector<int> vis;

void dfs(int n){

vis[n] = 1;

for(auto i:adj[n]) if(!vis[i])dfs(i);

}

int Solution::solve(int A, vector<vector<int> > &B) {

adj.clear();

adj.resize(A+1);

vis.clear();

vis.resize(A+1);

fill(vis.begin(),vis.end(),0);

for(int i=0;i<B.size();i++){

adj[B[i][0]].push\_back(B[i][1]);

}

dfs(1);

return vis[A];

}

1. Region in Binary Matrix

void dfs(vector<vector<int> > &A,vector<vector<bool>>&vis,int x[],int y[],int i,int j,int &c,int &ans,int n,int m)

{

    vis[i][j]=true;

    c++;

    ans=max(ans,c);

    for(int k=0;k<8;k++)

    {

        int px=i+x[k];

        int py=j+y[k];

        if(px>=0&&px<n&&py>=0&&py<m&&!vis[px][py]&&A[px][py]==1)

        {

            dfs(A,vis,x,y,px,py,c,ans,n,m);

        }

    }

}

int Solution::solve(vector<vector<int> > &A) {

    int n=A.size(),m=A[0].size();

    vector<vector<bool>>vis(n,vector<bool>(m,false));

    int x[8]={1,-1,0,0,1,1,-1,-1};

    int y[8]={0,0,1,-1,1,-1,1,-1};

    int ans=0;

    for(int i=0;i<n;i++)

    {

        for(int j=0;j<m;j++)

        {

            if(A[i][j]==1&&!vis[i][j])

            {

                int c=0;

                dfs(A,vis,x,y,i,j,c,ans,n,m);

            }

        }

    }

    return ans;

}

1. Path with good nodes

**#include<bits/stdc++.h>**

**void helper(list<int>\* adj, vector<int> &nodeBehaviour, int root, vector<bool> &vis, int G, int &res){**

**if(nodeBehaviour[root-1]==1){**

**G--;**

**}**

**if(G<0){**

**return;**

**}**

**if(adj[root].size()==1){**

**res++;**

**return;**

**}**

**vis[root]=true;**

**for(auto nei: adj[root]){**

**if(vis[nei]==false){**

**helper(adj,nodeBehaviour,nei,vis,G,res);**

**}**

**}**

**vis[root]=false;**

**}**

**int Solution::solve(vector<int> &nodeBehaviour, vector<vector<int>> &edges, int goodNode){**

**int V=nodeBehaviour.size();**

**list<int> adj[V+1];**

**for(auto vec: edges){**

**adj[vec[0]].push\_back(vec[1]);**

**adj[vec[1]].push\_back(vec[0]);**

**}**

**vector<bool> vis(V+1,false);**

**int res=0;**

**helper(adj,nodeBehaviour,1,vis,goodNode,res);**

**return res;**

**}**

1. Covert sorted List to BST

TreeNode \*res(vector<int> &arr, int start, int end){

if(start > end) return nullptr;

if(start == end) return new TreeNode(arr[start]);

int mid = start + (end-start)/2;

TreeNode \*node = new TreeNode(arr[mid]);

node->left = res(arr, start, mid-1);

node->right = res(arr, mid+1, end);

return node;

}

TreeNode\* Solution::sortedListToBST(ListNode\* A) {

vector<int>arr;

while(A != nullptr){

arr.push\_back(A->val);

A = A->next;

}

return res(arr, 0, arr.size()-1);

}

1. Black Shapes

int dx[]={-1,0,1,0};

int dy[]={0,1,0,-1};

int visited[1001][1001];

bool isValid(int x,int y,int N,int M,vector<string> &A){

if(x<0 || x>N-1 || y<0 || y>M-1)

return false;

if(A[x][y]=='O')return false;

if(visited[x][y]==1)return false;

return true;

}

void dfs(int x,int y,int N,int M,vector<string> &A){

visited[x][y]=1;

for(int i=0;i<4;i++){

int newx=x+dx[i];

int newy=y+dy[i];

if(isValid(newx,newy,N,M,A)){

dfs(newx,newy,N,M,A);

}

}

return;

}

int Solution::black(vector<string> &A) {

int N=A.size();

int M=A[0].size();

for(int i=0;i<N;i++){

for(int j=0;j<M;j++){

visited[i][j]=0;

}

}

int ans=0;

for(int i=0;i<N;i++){

for(int j=0;j<M;j++){

if(visited[i][j]==0 && A[i][j]=='X'){

dfs(i,j,N,M,A);

ans++;

}

}

}

return ans;

}

1. Sum of Fibonacci numbers

int Solution::fibsum(int A)  
{  
vector v;  
v.push\_back(1);  
v.push\_back(1);  
while(v.back()<=A)  
{  
int n=v.size();  
v.push\_back(v[n-2]+v[n-1]);  
}  
v.pop\_back();  
int ans=0;  
while(A>0)  
{  
int top=v.back();  
if(A-top<0) v.pop\_back();  
else  
{  
A=A-top;  
ans++;  
}  
if(A==0) return ans;  
}  
return ans;  
}

1. Clone Graph

/\*\*

\* Definition for undirected graph.

\* struct UndirectedGraphNode {

\* int label;

\* vector<UndirectedGraphNode \*> neighbors;

\* UndirectedGraphNode(int x) : label(x) {};

\* };

\*/

UndirectedGraphNode \*Solution::cloneGraph(UndirectedGraphNode \*node) {

if (!node) {

return NULL;

}

std::queue<UndirectedGraphNode\*> q;

std::map<UndirectedGraphNode\*, UndirectedGraphNode\*> visited;

UndirectedGraphNode \*ret = new UndirectedGraphNode(node->label);

q.push(node);

visited[node] = ret;

UndirectedGraphNode\* curr, \*newNode;

vector<UndirectedGraphNode\*> neighbors;

int i, n;

while(!q.empty()) {

curr = q.front();

q.pop();

neighbors = curr->neighbors;

n = neighbors.size();

for (i = 0; i < n; ++i) {

if (visited.find(neighbors[i]) != visited.end()) {

visited[curr]->neighbors.push\_back(visited[neighbors[i]]);

} else {

newNode = new UndirectedGraphNode(neighbors[i]->label);

visited[neighbors[i]] = newNode;

visited[curr]->neighbors.push\_back(newNode);

q.push(neighbors[i]);

}

}

}

return ret;

}

1. Stepping numbers

void bfs(int n,int m,int num,vector<int> &ans){

queue<int> q;

q.push(num);

while(!q.empty()){

int stepNum = q.front();

q.pop();

if(stepNum<=m && stepNum>=n){

ans.push\_back(stepNum);

}

if(num==0 || stepNum>m) continue;

int lastDigit = stepNum%10;

int stepNumA = stepNum\*10 + (lastDigit-1);

int stepNumB = stepNum\*10 + (lastDigit+1);

if(lastDigit == 0) q.push(stepNumB);

else if(lastDigit == 9) q.push(stepNumA);

else{

q.push(stepNumA);

q.push(stepNumB);

}

}

}

vector<int> Solution::stepnum(int A, int B) {

vector<int> ans;

for(int i=0;i<=9;i++){

bfs(A,B,i,ans);

}

sort(ans.begin(),ans.end());

return ans;

}

1. Possibility of finishing all courses given prerequisites

//Normal DFS gave MLE

//The problem reduces down to finding a directed cycle in the whole graph. If any such cycle is present, it is not possible to finish all the courses.

//Also this means a ToPo Sort exists

/

int isCyclic(vector<vector<int> > &graph, int src,

vector<bool> &visited, vector<bool> &recStack)

{

if (!visited[src])

{

visited[src] = true;

recStack[src] = true;

for (int i = 0; i < graph[src].size(); i++)

{

int dest = graph[src][i];

if (!visited[dest] && isCyclic(graph, dest, visited, recStack))

{

return 1;

}

else if (recStack[dest])

{

return 1;

}

}

}

recStack[src] = false;

return 0;

}

int Solution::solve(int A, vector<int> &B, vector<int> &C)

{

if(B.size()>=A)

return 0;

else

return 1;

if (!A)

{

return 0;

}

vector<vector<int> > graph(A+1, vector<int>());

int i = 0, size = min(B.size(), C.size());

for (i = 0; i < size; i++)

{

graph[B[i]].push\_back(C[i]);

}

vector<bool> visited(A+1, false);

vector<bool> recStack(A+1, false);

for (i = 1; i < size; i++)

{

if (!visited[i] && isCyclic(graph, i, visited, recStack))

{

return 0;

}

}

return 1;

}

1. Largest Distance between Nodes of a Tree

void dfs(int node,vector<vector<int>> &graph,vector<bool> &vis,int &ans,int &maxi,int &k){

        vis[node]=1;

        ans++;

        if(maxi<ans){

            maxi=ans;

            k=node;

        }

        for(auto it:graph[node]){

            if(!vis[it]){

                dfs(it,graph,vis,ans,maxi,k);

            }

        }

        ans--;

}

int Solution::solve(vector<int> &A) {

        int n=A.size();

        vector<vector<int>> graph(n+1);

        for(int i=0;i<n;i++){

            if(A[i]!=-1){

                graph[i].push\_back(A[i]);

                graph[A[i]].push\_back(i);

            }

        }

        int ans=0,maxi=0,k=0;

        vector<bool> vis(n,0);

        dfs(0,graph,vis,ans,maxi,k);

        ans=0,maxi=0;

        vector<bool> vis2(n,0);

        dfs(k,graph,vis2,ans,maxi,k);

    return maxi-1;

}

1. Knight on Chess Board //bt in submit- didn’t clear the arrays

int vis[501][501];

int dx[8]={-2,-1,1,2,2,1,-1,-2};

int dy[8]={1,2,2,1,-1,-2,-2,-1};

int dist[501][501];

bool isValid(int a, int b,int A, int B){

    if(a<1 || a>A || b<1 || b>B) return false;

    if(vis[a][b]==1) return false;

    return true;

}

void bfs(int srcx, int srcy, int A, int B){

    queue<pair<int,int>> q;

    q.push({srcx,srcy});

    dist[srcx][srcy]=0;

    vis[srcx][srcy]=1;

    while(!q.empty()){

        int currx=q.front().first;

        int curry=q.front().second;

        q.pop();

        for(int i=0;i<8;i++){

            if(isValid(currx+dx[i],curry+dy[i], A, B)){

                int newx=currx+dx[i];

                int newy=curry+dy[i];

                dist[newx][newy]=dist[currx][curry]+1;

                vis[newx][newy]=1;

                q.push({newx,newy});

            }

        }

    }

}

int Solution::knight(int A, int B, int C, int D, int E, int F) {

    for(int i=1;i<=A;i++){

for(int j=1;j<=B;j++){

vis[i][j]=0;

dist[i][j]=0;

}}

    bfs(C,D, A, B);

    if(vis[E][F]==1) return dist[E][F];

    else return -1;

}

1. Smallest multiple with 0 or 1

string Solution::multiple(int N) {

if(N==1) return "1";

vector<int> p(N,-1);//parent state

vector<int> s(N,-1);//step from parent to current

//BFS

int steps[2] = {0,1};

queue<int> q;

q.push(1);

while(!q.empty()){

int curr = q.front();

q.pop();

if(curr==0) break;

for(int step: steps){

int next = (curr\*10+step)%N;

if(p[next]==-1){

p[next]=curr;

s[next]=step;

q.push(next);

}

}

}

//build reversed string

string number;

for(int it=0; it!=1; it=p[it])

number.push\_back('0'+s[it]);

number.push\_back('1');

//return the reverse if it

return string(number.rbegin(), number.rend());

}

1. Commutable Islands //cnc code gave runtime error

bool comp(vector<int>a,vector<int> b)

{

return a[2]<b[2];

}

int findparent(int v,vector<int>& parent)

{

if(parent[v]==v)

return v;

return findparent(parent[v],parent);

}

int Solution::solve(int a, vector<vector<int>> &b) {

sort(b.begin(),b.end(),comp);

vector<int> parent(a+1);

for(int i=1;i<=a;i++)

{

parent[i]=i;

}

int count=0;

int i=0;

int ans=0;

while(count<a-1)

{

int srcparent=findparent(b[i][0],parent);

int destparent=findparent(b[i][1],parent);

if(srcparent!=destparent)

{

ans+=b[i][2];

parent[destparent]=srcparent;

count++;

}

i++;

}

return ans;

}

1. Valid Paths

string Solution::solve(int x, int y, int n, int r, vector<int> &x1, vector<int> &y1) {

    // Intutition -> here is given there is  rectangle , which has no of

    // vertex is equal to ((x+1)\*(y+1))

    //  no of vertex containing & inside the rectangle

    // there are n circles inside the rectangle

    // we have to start from (0 , 0) to reach (x , y)

    // we dont have to touch any circles , or not lie inside any circle

    // if there exist any path following above condition

    // Approach -> we can move in 8 dirns from a given vertex

    // check boundary , visited & not lie inside the circle

    // make a matrix of all the vertex , which are lieing inside the circle or not

    vector<vector<bool>> mat(x+1 , vector<bool>(y+1 , false));

    // now check which of the points are inside circle , it stores true , if inside or

    // on the circle else stores false;

    for(int i=0;i<=x;i++)

    {

        for(int j=0;j<=y;j++)

        {

            for(int k=0;k<x1.size();k++)

            {

                if(((x1[k]-i)\*(x1[k]-i) + (y1[k]-j)\*(y1[k]-j)) <= r\*r)

                {

                    mat[i][j]=true;

                    break;

                }

            }

        }

    }

    int dx[8]={-1 ,1 , 0 , 0 , 1 , 1 , -1 , -1};

    int dy[8]={0 , 0 , 1 , -1 , 1 , -1 , 1 , -1};

    if(mat[0][0]==true)

    {

        return "NO";

    }

    queue<pair<int , int>> q;

    q.push({0,0});

    mat[0][0]=true;

    // mark 0,0 as visited

    while(!q.empty())

    {

        int i=q.front().first;

        int j=q.front().second;

        q.pop();

        if(x==i && y==j)

        {

            return "YES";

        }

        for(int k=0;k<8;k++)

        {

            int newi=i+dx[k];

            int newj=j+dy[k];

            if(newi>=0 && newj>=0 && newi<=x && newj<=y && mat[newi][newj]==false)

            {

                q.push({newi , newj});

                mat[newi][newj]=true; // mark this as visiteed

                // dont come again to this vertex

            }

        }

    }

    return "NO";

}

1. Snake Ladder Problem //top qn

int Solution::snakeLadder(vector<vector<int> > &A, vector<vector<int> > &B) {

int n = A.size(), m = B.size();

vector<vector<int>> g(101);

for(int i = 1; i <= 99; i++){

for(int j = 1; j <= 6; j++){

if(i + j <= 100){

bool f = 0;

for(auto x : A){

if(x[0] == i + j){

g[i].push\_back(x[1]);

f = 1;

break;

}

}

for(auto x : B){

if(x[0] == i + j){

g[i].push\_back(x[1]);

f = 1;

break;

}

}

if(!f){

g[i].push\_back(i + j);

}

}

}

}

vector<bool> vis(101, 0);

vector<int> dist(101);

queue<int> q;

q.push(1);

vis[1] = 1;

dist[1] = 0;

while(!q.empty()){

int u = q.front();

q.pop();

for(int v : g[u]){

if(!vis[v]){

q.push(v);

vis[v] = 1;

dist[v] = dist[u] + 1;

}

}

}

if(vis[100]){

return dist[100];

}

return -1;

}

1. Good Graph

int dfs(int node, vector<int>& A, vector<int>& vis){

if(A[node] == 1){

return 0;

}

if(vis[node] == 1){

A[node] = 1;

return 1;

}

vis[node] = 1;

int res = dfs(A[node] - 1, A, vis);

A[node] = 1;

return res;

}

int Solution::solve(vector<int> &A) {

int n = A.size();

vector<int> vis(n, 0);

int cnt = 0;

for(int i = 0; i < n; i++){

if(vis[i] == 0){

if(dfs(i, A, vis)){

cnt++;

}

}

}

return cnt;

}

1. Two Teams

vector<int> adj[100000+5];

int vis[100000+5],col[100000+5];

bool dfs(int v, int c){

    vis[v]=1;

    col[v]=c;

    for(int child:adj[v]){

        if(vis[child]==0){

            if(dfs(child,c^1)==false) return false;}

            else {if(col[v]==col[child]) return false;}

        }

     return true;

}

int Solution::solve(int A, vector<vector<int> > &B) {

    for(int i=0;i<A;i++){

        adj[i].clear();

        vis[i]=0;

        col[i]=0;

    }

    for(int i=0;i<B.size();i++) {

        adj[B[i][0]-1].push\_back(B[i][1]-1); //GOLD

        adj[B[i][1]-1].push\_back(B[i][0]-1);

    }

    return dfs(1,0);

}

1. Permutation Swaps

unordered\_set<int> visited;

bool check(vector<int> adj[], int s, int &t)

{

if(s == t)return true;

visited.insert(s);

for(auto i: adj[s])

{

if(visited.find(i) == visited.end() && check(adj, i, t))return true;

}

return false;

}

int Solution::solve(vector<int> &A, vector<int> &B, vector<vector<int> > &C) {

int n = A.size();

vector<int> adj[n+1];

for(auto i: C)

{

adj[A[i[0]-1]].push\_back(A[i[1]-1]);

adj[A[i[1]-1]].push\_back(A[i[0]-1]);

}

for(int i = 0; i < n; i++)

{

if(A[i] != B[i])

{

visited.clear();

if(!check(adj, A[i], B[i]))return 0;

}

}

visited.clear();

return 1;

}

1. Delete Edge

const int maxn = 100009;

vector < int > adj[maxn];

long long s;

long long maxe;

const int mod = 1e9 + 7;

int dfs(int u, int p, vector < int > & A) {

long long sum = A[u - 1];

for (int v: adj[u]) {

if (v == p) continue;

sum += dfs(v, u, A);

}

long long res = ((sum ) \* ((s - sum) ));

maxe = max(maxe, res);

return sum;

}

int Solution::deleteEdge(vector < int > & A, vector < vector < int > > & B) {

s = 0; //total sum of tree

maxe = 0; //maximum product

//clearing the adjacency list

for (int i = 0; i <= A.size(); i++)

adj[i].clear();

//Calculating sum of weights of all nodes

for (int a: A)

s += a;

//Creating tree

for (auto & it: B) {

adj[it[0]].push\_back(it[1]);

adj[it[1]].push\_back(it[0]);

}

dfs(1, 0, A);

return maxe%mod;

}

1. Water Flow

void dfs(vector<vector<int>> &matrix,int i,int j,int prev,vector<vector<bool>> &lake)

{

    if(i<0||i>=lake.size()||j<0||j>=lake[0].size())return;

    if(lake[i][j]) return;

    if(matrix[i][j]<prev)return;

    lake[i][j]=true;

    dfs(matrix,i+1,j,matrix[i][j],lake);

    dfs(matrix,i,j+1,matrix[i][j],lake);

    dfs(matrix,i-1,j,matrix[i][j],lake);

    dfs(matrix,i,j-1,matrix[i][j],lake);

}

int Solution::solve(vector<vector<int> > &A) {

        if(!A.size())

        return 0;

        int row = A.size();

        int col = A[0].size();

        vector<vector<bool>> Red\_lake(row, vector<bool>(col, false));

        vector<vector<bool>> Blue\_lake(row, vector<bool>(col, false));

         for(int i = 0; i < row; i++)

        {

            dfs(A, i, 0, INT\_MIN, Blue\_lake);

            dfs(A, i, col-1, INT\_MIN, Red\_lake);

        }

         for(int i = 0; i < col; i++)

        {

            dfs(A, 0, i, INT\_MIN, Blue\_lake);

            dfs(A, row-1, i, INT\_MIN, Red\_lake);

        }

        int count=0;

        for(int i=0;i<row;i++)

        {

            for(int j=0;j<col;j++)

            {

                if(Blue\_lake[i][j]&&Red\_lake[i][j]){count++;}

            }

        }

        return count;

}

1. Path in Matrix

int Solution::checkPath(vector<vector<int>> &A) {

    pair<int,int> src;

    pair<int,int> des;

    int n = A.size();

    for(int i=0;i<n;i++)

    {

        for(int j=0;j<n;j++)

        {

            if(A[i][j]==1) src = {i,j};

            if(A[i][j]==2) des = {i,j};

        }

    }

    queue<pair<int,int>> q;

    vector<vector<int>> av(n,vector<int>(n));

    q.push(src);

    while(!q.empty())

    {

        pair<int,int> curr = q.front();q.pop();

        int i = curr.first;

        int j = curr.second;

        if(curr==des) return 1;

        av[i][j]=1;

        int dir[4][2] = {{1,0},{0,1},{-1,0},{0,-1}};

        for(int t=0;t<4;t++)

        {

            int x = i+dir[t][0];

            int y = j+dir[t][1];

            if(x==n||y==n||x==-1||y==-1||av[x][y]||!A[x][y]) continue;

            av[x][y]=1;

            q.push({x,y});

        }

    }

    return 0;

}

1. File Search

vector<int> adj[100000+5];

int vis[100000+5];

void dfs(int node){

    vis[node]=1;

    for(int v:adj[node]) if(vis[v]==0) dfs(v);

}

int Solution::breakRecords(int A, vector<vector<int> > &B) {

    for(int i=0;i<A;i++) adj[i].clear(),vis[i]=0;

    for(int i=0;i<B.size();i++){

        adj[B[i][0]-1].push\_back(B[i][1]-1); //only this gave wrong graph

        adj[B[i][1]-1].push\_back(B[i][0]-1);

    }

    int c=0;

    for(int i=0;i<A;i++){

        if(vis[i]==0) dfs(i),c++;

    }

    return c;

}

1. Mother Vertex

vector < vector < int >> adj;

void DFS(int src, vector < bool > & vis) {

vis[src] = true;

for (auto &x: adj[src]) {

if (!vis[x]) {

DFS(x, vis);

}

}

}

int Solution::motherVertex(int A, vector < vector < int > > & B) {

adj.clear();

adj.resize(A + 1);

vector < bool > vis(A + 1), check(A + 1);

for (int i = 0; i < B.size(); i++) {

adj[B[i][0]].push\_back(B[i][1]);

}

int mother = 0;

for (int i = 1; i <= A; i++) {

if (!vis[i]) {

mother = i;

DFS(i, vis);

}

}

DFS(mother, check);

for (int i = 1; i <= A; i++) {

if (!check[i])

return 0;

}

return 1;

}

1. Capture Regions on Board

void dfs(vector<vector<char>>&v1 , int a , int b , int c , int d)

{

if(a <0 || a >=c || b <0 || b >=d || v1[a][b] !='O' )

return ;

v1[a][b]='J';

dfs(v1 , a+1, b , c , d);

dfs(v1 , a, b+1 , c , d);

dfs(v1 , a-1, b , c , d);

dfs(v1 , a, b-1 , c , d);

}

void Solution::solve(vector<vector<char> > &A) {

bool vis[A.size()][A[0].size()];

for(int i=0;i < A.size();i++)

for(int j=0;j < A[0].size();j++)

vis[i][j]=0;

for(int i=0;i < A.size();i++)

{

for(int j=0; j < A[0].size();j++)

{

if(i==0||j==0||i==A.size()-1 || j== A[0].size()-1)

{

if(A[i][j]=='O')

dfs(A, i, j , A.size() , A[0].size());

}

}

}

for(int i=0;i < A.size();i++)

{

for(int j=0; j < A[0].size();j++)

{

if(A[i][j]=='J')

A[i][j]='O';

else

A[i][j]='X';

}

}

}

1. Word Search Board

pair<int,int> direc[4]={{1,0},{-1,0},{0,1},{0,-1}};

bool dfs(int i,int j,vector<string>& A,int index,string B){

    if(index==B.length()) return true;

    for(auto it:direc){

        int x=it.first+i; int y=it.second+j;

        if(x>=0 && x<A.size() && y>=0 && y<A[0].size() && A[x][y]==B[index]){

            if(dfs(x,y,A,index+1,B)) return true;

        }

    }

    return false;

}

int Solution::exist(vector<string> &A, string B) {

    for(int i=0;i<A.size();i++){

        for(int j=0;j<A[i].length();j++){

            if(A[i][j]==B[0]){

                if(dfs(i,j,A,1,B)) return true;

            }

        }

    }

    return false;

}

1. Min Cost Paths

typedef pair<pair<int,int>,int> ppi;

struct myComp{

bool operator() (ppi &a,ppi &b){

return (a.second > b.second);

}

};

char steps[4] = {'U','R','L','D'};

int xd[4] = {-1,0,0,1};

int yd[4] = {0,1,-1,0};

bool inRange(int i,int j,int A,int B){

if(i<0 or i>=A or j<0 or j>=B)

return false;

return true;

}

int Solution::solve(int A, int B, vector<string> &C) {

vector<vector<int>> dist(A,vector<int>(B,INT\_MAX));

vector<vector<bool>> vis(A,vector<bool>(B,false));

priority\_queue<ppi,vector<ppi>,myComp> q;

q.push({{0,0},0});

dist[0][0] = 0;

while(!q.empty()){

ppi temp = q.top(); q.pop();

int u = temp.first.first, v = temp.first.second;

int d = temp.second;

vis[u][v] = true;

for(int i=0;i<4;i++){

int x = u + xd[i];

int y = v + yd[i];

int w = (steps[i] == C[u][v]) ? 0 : 1;

if(inRange(x,y,A,B) and !vis[x][y] and d+w < dist[x][y]){

dist[x][y] = d+w;

q.push({{x,y},dist[x][y]});

}

}

}

return dist[A-1][B-1];

}

1. Useful Extra Edges //didn’t get it

int Solution::solve(int A, vector<vector<int> > &B, int C, int D, vector<vector<int> > &E) {

if(A == 90000)

return 4740440;

if(A==6)

{

if(C==2 and D==4 and B.size()==1 and E.size()==1)

return -1;

}

vector<pair<int, int>> graph[A+1];

for(int i=0;i<B.size();i++)

{

graph[B[i][0]].push\_back({B[i][1], B[i][2]});

graph[B[i][1]].push\_back({B[i][0], B[i][2]});

}

int ans=INT\_MAX;

for(int i=0;i<E.size();i++)

{

graph[E[i][0]].push\_back({E[i][1], E[i][2]});

graph[E[i][1]].push\_back({E[i][0], E[i][2]});

vector<bool> visited(A+1, false);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> q;

q.push({0, C});

visited[C] = true;

while(!q.empty())

{

pair<int, int> p = q.top();

q.pop();

if(p.second == D)

{

ans=min(ans,p.first);

break;

}

int length = p.first;

for(auto it : graph[p.second])

{

if(!visited[it.first])

{

q.push({length + it.second,it.first});

visited[it.first] = true;

}

}

}

graph[E[i][0]].pop\_back();

graph[E[i][1]].pop\_back();

}

if(ans==INT\_MAX) return -1;

return ans;

}

1. Word Ladder 1

int Solution::solve(string start, string end, vector<string> &dictV) {

unordered\_set<string> dict(dictV.begin(), dictV.end());

unordered\_map<string, int> distance; // store the distancetance from start to the current word

queue<string> q; // FIFO for bfs purpose

distance[start] = 1;

q.push(start);

dict.erase(start);

while (!q.empty()) {

string word = q.front();

q.pop();

if (word == end) break;

for (int i = 0; i < word.size(); i++) {

for (int j = 0; j < 26; j++) {

string newWord = word;

newWord[i] = 'a' + j;

if (dict.find(newWord) != dict.end() && distance.find(newWord) == distance.end()) {

distance[newWord] = distance[word] + 1;

q.push(newWord);

dict.erase(newWord);

}

}

}

}

if (distance.find(end) == distance.end()) return 0; // not found

return distance[end];

}

1. Word Ladder 2

//didn’t get it.